Audio classification using machine learning

Data: https://urbansounddataset.weebly.com/download-urbansound8k.html (https://urbansounddataset.weebly.com/download-urbansound8k.html)

In [1]:
```
# !pip install librosa
```

In [2]:
```
import numpy as np

```

In [3]:
```
import tensorflow as tf
```

In [4]:
```
try:
    import librosa
    print("Librosa is available.")
except ImportError:
    print("Librosa is not installed.")

```

```
Librosa is available.
```

In [5]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

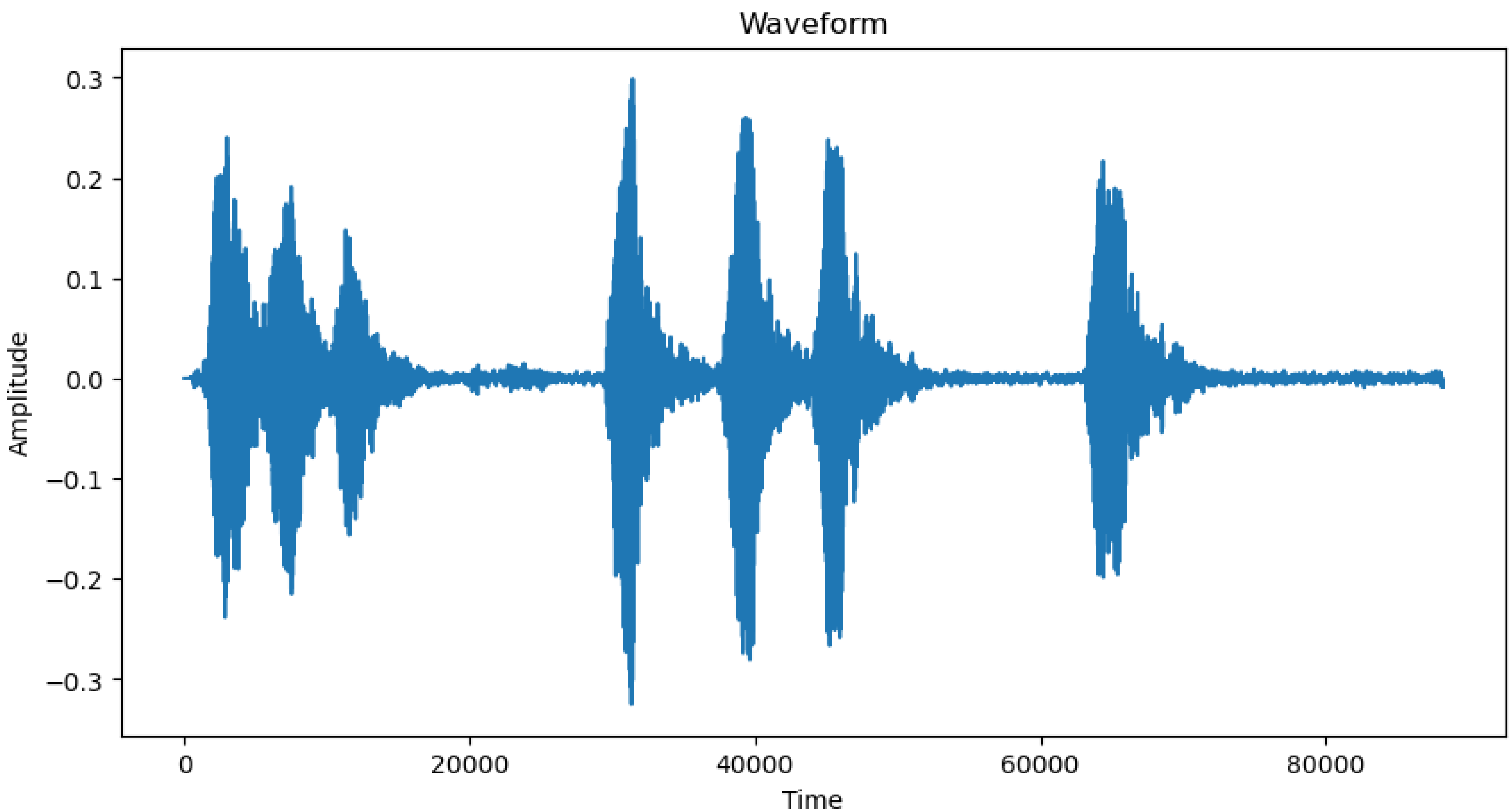
In [6]:
```
import IPython.display as ipd
import librosa
import librosa.display
```

In [7]:
```
filename = 'dog bark.wav'
```

In [8]:

```
data, sample_rate = librosa.load(filename)
plt.figure(figsize=(10, 5))
plt.plot(data)
plt.title('Waveform')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()

ipd.Audio(filename)
```



Out[8]:

0:04 / 0:04

In [9]:
```
sample_rate
```
Out[9]:
```
22050
```

In [10]:
```
from scipy.io import wavfile as wav
wave_sample_rate, wave_audio = wav.read(filename)
```

In [11]:
```
wave_sample_rate
```
Out[11]:
```
44100
```

In [12]:
```
wave_audio
```
Out[12]:
```
array([[   0,    0],
       [   0,    0],
       [   0,    0],
       ...,
       [-399, -115],
       [-388, -111],
       [-386, -105]], dtype=int16)
```

In [13]:
```
data
```
Out[13]:
```
array([ 3.4924597e-10,  3.4924597e-10,  4.6566129e-10, ...,
       -7.9498515e-03, -7.7366987e-03, -8.0531817e-03], dtype=float32)
```

In [14]:
```
import pandas as pd

metadata = pd.read_csv('UrbanSound8K/metadata/UrbanSound8K.csv')
```

In [15]:
```
metadata.head(10)
```

Out[15]:

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
5	100263-2-0-143.wav	100263	71.500000	75.500000	1	5	2	children_playing
6	100263-2-0-161.wav	100263	80.500000	84.500000	1	5	2	children_playing
7	100263-2-0-3.wav	100263	1.500000	5.500000	1	5	2	children_playing
8	100263-2-0-36.wav	100263	18.000000	22.000000	1	5	2	children_playing
9	100648-1-0-0.wav	100648	4.823402	5.471927	2	10	1	car_horn

In [16]:
```
# check wether the dataset is imbalanced
metadata['class'].value_counts()
```

Out[16]:
```
dog_bark            1000
children_playing    1000
air_conditioner     1000
street_music        1000
engine_idling       1000
jackhammer          1000
drilling            1000
siren                929
car_horn             429
gun_shot             374
Name: class, dtype: int64
```

In [17]:
```
len(metadata)
```

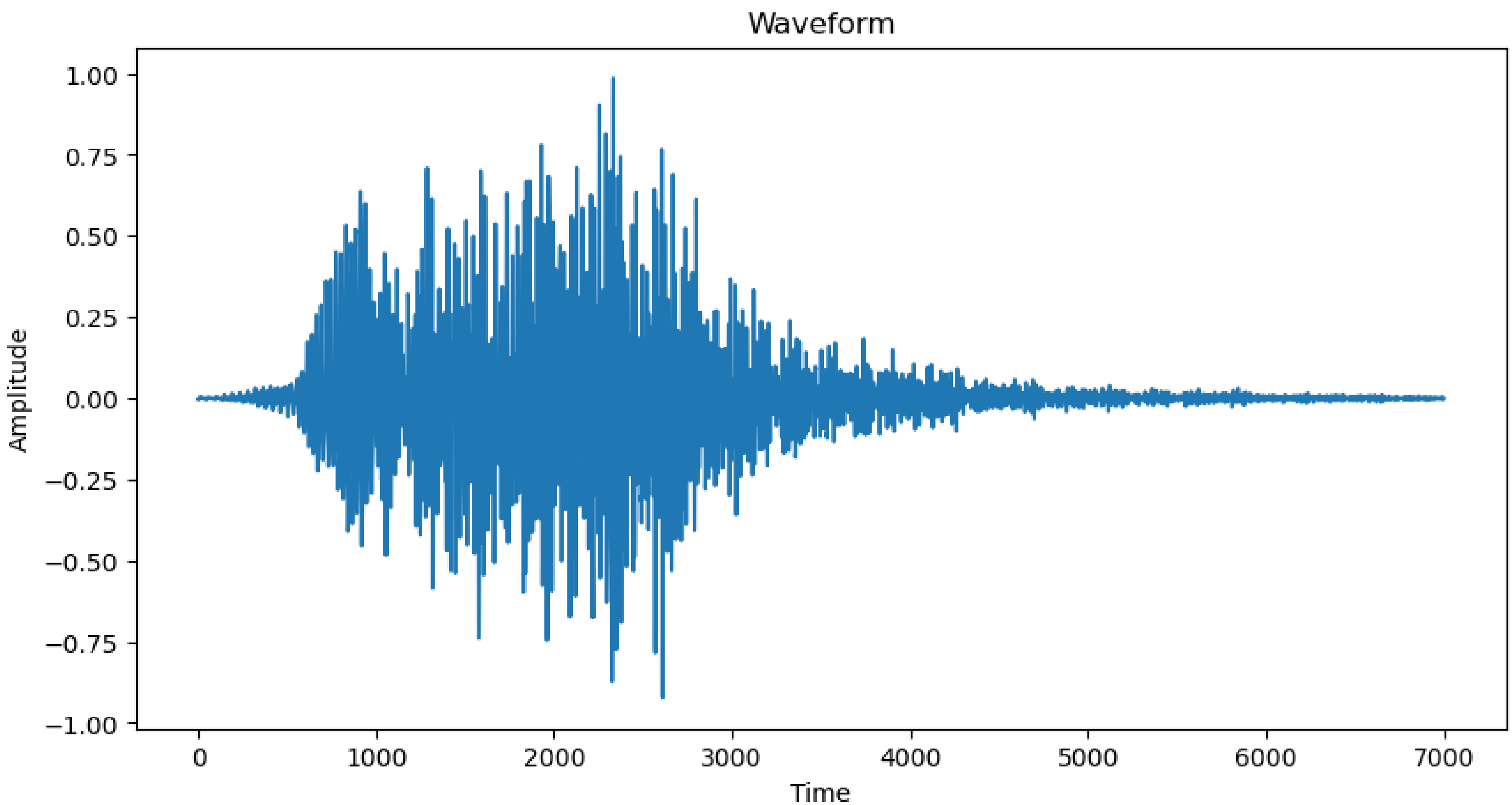
Out[17]:
```
8732
```

In [18]:
```
metadata.shape
```

Out[18]:
```
(8732, 8)
```

In [19]:
```
filename2 = 'dog bark2.wav'
data, sample_rate = librosa.load(filename2)
plt.figure(figsize=(10, 5))
plt.plot(data)
plt.title('Waveform')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()

ipd.Audio(filename2)
```



Out[19]:

0:00 / 0:00

Preprocessing data

In [20]:
```
audio_file_path = '100263-2-0-3.wav'
librosa_audio_data, librosa_sample_rate = librosa.load(audio_file_path)
```

In [21]:
```
print(librosa_audio_data)
```

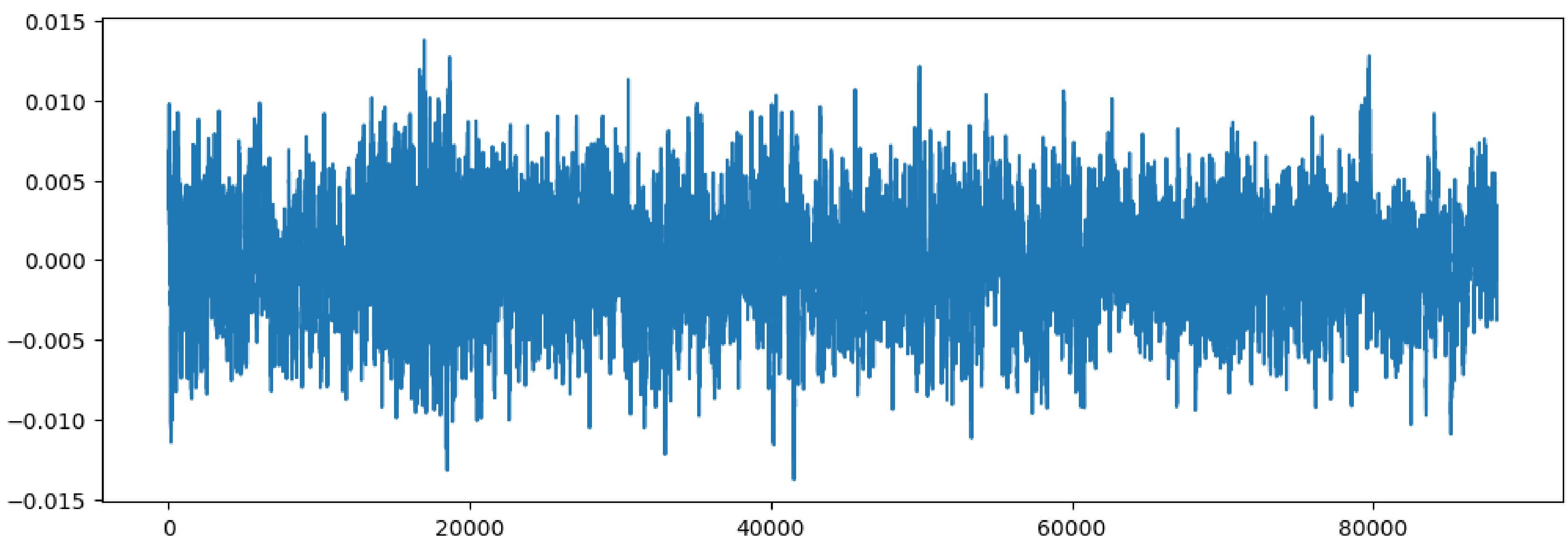
```
[ 0.00331575  0.00467553  0.00361099 ... -0.00376796 -0.00347471
 -0.00357828]
```

Observation

Here Librosa converts the signal to mono, meaning the channel array will be 1:

[0.00331575 0.00467553 0.00361099 ... -0.00376796 -0.00347471 -0.00357828]

In [22]:
```
# let plot the librosa audio data
# original audio with 1 channel
plt.figure(figsize = (12, 4))
plt.plot(librosa_audio_data);
```



In [23]:
```
# let's read with scipy
wave_sample_rate, wave_audio = wav.read(audio_file_path)
```

Here we have 2 channels

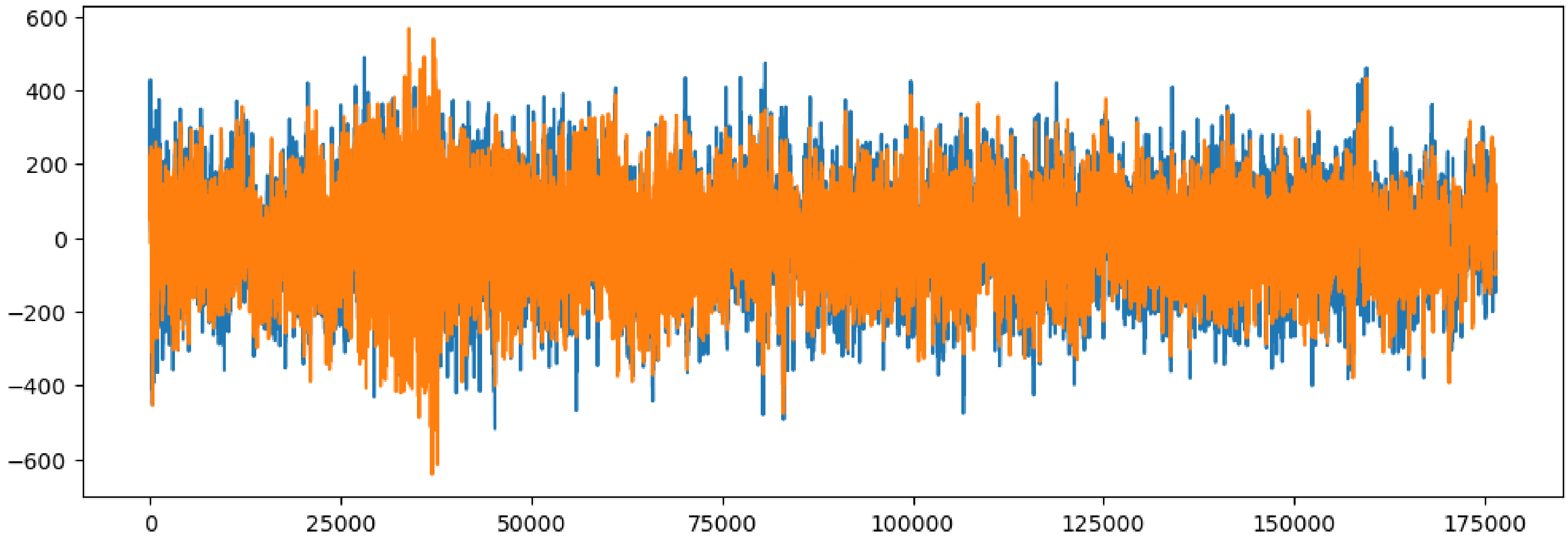
(array([[194, 100], [179, 113], [160, 124], ..., [-143, -87], [-134, -91], [-110, -98]], dtype=int16)

In [24]:
```
wave_audio, wave_sample_rate
```

Out[24]:
```
(array([[ 194,  100],
        [ 179,  113],
        [ 160,  124],
        ...,
        [-143,  -87],
        [-134,  -91],
        [-110,  -98]], dtype=int16),
 44100)
```

In [25]:
```
# plot Original audio with 2 channels

plt.figure(figsize=(12, 4))
plt.plot(wave_audio);
```



Extracting Features

Here we will be using Mel_Frequency Cepstral Coefficient(MFCC) from the audio samples. The MFCC summarises the frequency distribution across the window size, so it is ossible to analyse both the fregquency and time characteristics of the sound. These audio representation will allow us to identify features for classification.

https://chat.openai.com/share/53e7a8f8-90fc-489c-9b4e-e6584c1b02c5 (https://chat.openai.com/share/53e7a8f8-90fc-489c-9b4e-e6584c1b02c5)

In [26]:
```
mfccs = librosa.feature.mfcc(y = librosa_audio_data, sr = librosa_sample_rate, n_mfcc=40)
```

In [27]:
```
mfccs
```

Out[27]:
```
array([[-4.7486273e+02, -4.5088608e+02, -4.4905338e+02, ...,
        -4.7676157e+02, -4.7334869e+02, -4.9085260e+02],
       [ 1.1530264e+02,  1.1144249e+02,  1.1125224e+02, ...,
         1.1112500e+02,  1.1057970e+02,  1.0299150e+02],
       [-1.8326149e+01, -2.4682453e+01, -3.0259779e+01, ...,
        -8.2357616e+00, -9.0665359e+00, -4.5019574e+00],
       ...,
       [-2.8760438e+00, -3.2479770e+00, -4.8965530e+00, ...,
        -5.2023709e-01,  3.5672512e+00,  7.4937592e+00],
       [-4.2968428e-01, -5.8838773e-01, -8.1724101e-01, ...,
         1.8340731e-01,  7.6732409e-01,  2.7120016e+00],
       [-1.1780634e+00,  6.9809389e-01,  6.3521605e+00, ...,
        -2.6221924e+00, -4.7912717e+00, -3.1826315e+00]], dtype=float32)
```

In [28]:
```
print(mfccs.shape)
```

```
(40, 173)
```

Extracting MFCC's for every audio file

In [29]:
```
# pip install resampy

```

In [30]:
```
import os
import numpy as np

```

In [31]:
```
audio_dataset_path = 'UrbanSound8K/audio/'
metadata = pd.read_csv('UrbanSound8K/metadata/UrbanSound8K.csv')
metadata.head()
```

Out[31]:

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

In [32]:
```
import os
import librosa
import numpy as np
from tqdm import tqdm

# Creating a function to extract scaled MFCC features
def features_extractor(file_name):
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)
    return mfccs_scaled_features
```

In [33]:

```python
# Iterate through every audio file and extract features using Mel-frequency cepstral coefficients
extracted_features = []
for index_num, row in tqdm(metadata.iterrows()):
    file_name = os.path.join(os.path.abspath(audio_dataset_path), 'fold'+ str(row['fold'])+'/', str(r
    final_class_labels = row['class']
    data = features_extractor(file_name)
    extracted_features.append([data, final_class_labels])
```

```
3555it [11:01,  5.59it/s]C:\Users\USER\anaconda3\lib\site-packages\librosa\core\spectrum.py:256: UserWar
ning: n_fft=2048 is too large for input signal of length=1323
  warnings.warn(
8326it [24:59,  8.48it/s]C:\Users\USER\anaconda3\lib\site-packages\librosa\core\spectrum.py:256: UserWar
ning: n_fft=2048 is too large for input signal of length=1103
  warnings.warn(
8328it [24:59, 10.57it/s]C:\Users\USER\anaconda3\lib\site-packages\librosa\core\spectrum.py:256: UserWar
ning: n_fft=2048 is too large for input signal of length=1523
  warnings.warn(
8732it [26:43,  5.44it/s]
```

In [81]:

```python
extracted_features
```

Out[81]:

```
[[array([-217.35526   ,   70.22338   , -130.38527   ,  -53.282898  ,
          -21.199127  ,  -22.677624  ,  -10.85597   ,   18.294254  ,
            6.6527047 ,   14.324022  ,  -12.167681  ,    2.2768362 ,
          -17.779186  ,   10.388948  ,   -6.582835  ,   -0.6944583 ,
          -18.336023  ,    1.9942524 ,   -5.1433306 ,    8.3024    ,
          -12.645056  ,   -6.5297318 ,    4.617668  ,   -2.1799176 ,
           -6.662823  ,    0.35971084,   -3.908409  ,    4.775624  ,
           -6.3845205 ,   -5.379818  ,    0.9159791 ,    6.970493  ,
           -0.24866596,    1.6782186 ,   -5.6111803 ,   -2.9643464 ,
            3.1490579 ,   -1.6930529 ,   -0.6169833 ,    0.38600507],
        dtype=float32),
  'dog_bark'],
 [array([-4.2409818e+02,  1.0934077e+02, -5.2919525e+01,  6.0864750e+01,
          2.4529217e-01,  1.7347328e+01,  2.0955825e+00,  1.0712966e+01,
         -1.3986129e+00,  1.2310798e+01, -1.1208863e+01,  2.1075325e+01,
         -9.4902792e+00,  1.3526470e+01, -2.3910540e-01,  7.1590004e+00,
         -3.0809760e+00,  1.1367644e+01, -6.3027663e+00,  6.8781152e+00,
         -2.8902097e+00,  8.7614346e+00, -2.3304422e+00,  8.4166384e+00,
         -1.6700815e+00,  1.2775006e+00, -4.4198775e+00,  1.2626288e+00,
```

In [82]:
```
len(extracted_features)
```

Out[82]:
```
8732
```

In [83]:
```
extracted_features_df = pd.DataFrame(extracted_features, columns=['feature', 'class'])
extracted_features_df.head()
```

Out[83]:

	feature	class
0	[-217.35526, 70.22338, -130.38527, -53.282898,...	dog_bark
1	[-424.09818, 109.34077, -52.919525, 60.86475, ...	children_playing
2	[-458.79114, 121.38419, -46.52066, 52.00812, -...	children_playing
3	[-413.89984, 101.66371, -35.42945, 53.036354, ...	children_playing
4	[-446.60352, 113.68541, -52.402214, 60.302044,...	children_playing

In [84]:
```
# Split the dataset into independent and dependednt dataset
x = np.array(extracted_features_df['feature'].tolist())
y = np.array(extracted_features_df['class'].tolist())
```

In [85]:
```
x
```

Out[85]:
```
array([[-2.1735526e+02,  7.0223381e+01, -1.3038527e+02, ...,
        -1.6930529e+00, -6.1698329e-01,  3.8600507e-01],
       [-4.2409818e+02,  1.0934077e+02, -5.2919525e+01, ...,
         5.3489321e-01, -5.4468715e-01,  4.4632098e-01],
       [-4.5879114e+02,  1.2138419e+02, -4.6520660e+01, ...,
         2.0768483e+00,  1.6962965e+00, -9.6140963e-01],
       ...,
       [-3.0388824e+02,  1.1135945e+02, -4.5941566e+01, ...,
        -3.0292377e+00,  2.7170296e+00,  7.6197419e+00],
       [-3.4411008e+02,  1.2545021e+02, -5.4903442e+01, ...,
        -7.9082427e+00, -1.6414584e+00,  5.6668439e+00],
       [-3.1560281e+02,  9.4854805e+01, -3.7222340e+01, ...,
         6.1386460e-01, -1.1449189e+01, -6.0105853e+00]], dtype=float32)
```

In [86]:
```
y
```
Out[86]:
```
array(['dog_bark', 'children_playing', 'children_playing', ...,
       'car_horn', 'car_horn', 'car_horn'], dtype='<U16')
```

In [87]:
```
x.shape, y.shape
```
Out[87]:
```
((8732, 40), (8732,))
```

In [89]:
```
# label encoding (converting it ti numbers)
# y = np.array(pd.get_dummies(y))
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
y = to_categorical(labelencoder.fit_transform(y))
```

In [90]:
```
y
```
Out[90]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

In [91]:
```
y.shape
```
Out[91]:
```
(8732, 10)
```

In [92]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

In [93]:
```
x_train
```

Out[93]:
```
array([[-1.3110471e+02,  1.1250591e+02, -2.2574696e+01, ...,
          3.2466526e+00, -1.3690237e+00,  2.7557542e+00],
       [-1.3670342e+01,  9.1085083e+01, -7.7927337e+00, ...,
         -3.2530508e+00, -5.2774529e+00, -1.5569715e+00],
       [-4.9871544e+01,  2.6535299e-01, -2.0500937e+01, ...,
          2.8545945e+00, -1.6092046e+00,  3.5248058e+00],
       ...,
       [-4.2701236e+02,  9.2623047e+01,  3.1293974e+00, ...,
          7.4264139e-01,  7.3349088e-01,  7.1100914e-01],
       [-1.4575461e+02,  1.3626578e+02, -3.3515518e+01, ...,
          1.4681193e+00, -2.0091701e+00, -8.8218188e-01],
       [-4.2103134e+02,  2.1065454e+02,  3.4906609e+00, ...,
         -5.3888674e+00, -3.3713605e+00, -1.5665115e+00]], dtype=float32)
```

In [94]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[94]:
```
((6985, 40), (6985, 10), (1747, 40), (1747, 10))
```

Model creation

In [95]:
```
import tensorflow as tf
import keras

print("TensorFlow version:", tf.__version__)
print("Keras version:", keras.__version__)
```

```
TensorFlow version: 2.12.0
Keras version: 2.12.0
```

In [96]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn import metrics

```

In [97]:
```
# number of classes
num_labels = y.shape[1]
```

In [98]:
```
Dense(units=32, activation='relu')

```

Out[98]:
```
<keras.layers.core.dense.Dense at 0x1ffb539dc40>
```

In [99]:
```
model = Sequential()

# first layer
model.add(Dense(100, input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# second layer
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# third layer
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# final layer
model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

In [100]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_6 (Dense)             (None, 100)               4100      
                                                                 
 activation_4 (Activation)   (None, 100)               0         
                                                                 
 dropout_3 (Dropout)         (None, 100)               0         
                                                                 
 dense_7 (Dense)             (None, 200)               20200     
                                                                 
 activation_5 (Activation)   (None, 200)               0         
                                                                 
 dropout_4 (Dropout)         (None, 200)               0         
                                                                 
 dense_8 (Dense)             (None, 100)               20100     
                                                                 
 activation_6 (Activation)   (None, 100)               0         
                                                                 
 dropout_5 (Dropout)         (None, 100)               0         
                                                                 
 dense_9 (Dense)             (None, 10)                1010      
                                                                 
 activation_7 (Activation)   (None, 10)                0         
                                                                 
=================================================================
Total params: 45,410
Trainable params: 45,410
Non-trainable params: 0
_________________________________________________________________
```

In [101]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [102]:

```
# Training our model
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 100
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='model/audio_classification.hdf5', verbose=1, save_best_only=
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(x_test, y_

duration = datetime.now() - start
print('Training completed in time:', duration)

```

```
210/219 [===========================>..] - ETA: 0s - loss: 0.9477 - accuracy: 0.6851
Epoch 97: val_loss improved from 0.74097 to 0.73980, saving model to model\audio_classification.hdf5
219/219 [==============================] - 2s 8ms/step - loss: 0.9511 - accuracy: 0.6842 - val_loss:
0.7398 - val_accuracy: 0.7705
Epoch 98/100
212/219 [============================>.] - ETA: 0s - loss: 0.9590 - accuracy: 0.6891
Epoch 98: val_loss improved from 0.73980 to 0.73447, saving model to model\audio_classification.hdf5
219/219 [==============================] - 2s 9ms/step - loss: 0.9584 - accuracy: 0.6885 - val_loss:
0.7345 - val_accuracy: 0.7768
Epoch 99/100
218/219 [============================>.] - ETA: 0s - loss: 0.9358 - accuracy: 0.6892
Epoch 99: val_loss did not improve from 0.73447
219/219 [==============================] - 1s 6ms/step - loss: 0.9358 - accuracy: 0.6892 - val_loss:
0.7430 - val_accuracy: 0.7687
Epoch 100/100
212/219 [============================>.] - ETA: 0s - loss: 0.9290 - accuracy: 0.6862
Epoch 100: val_loss did not improve from 0.73447
219/219 [==============================] - 1s 6ms/step - loss: 0.9281 - accuracy: 0.6875 - val_loss:
0.7567 - val_accuracy: 0.7710
Training completed in time: 0:02:47.064258
```

In [103]:

```
test_accuracy = model.evaluate(x_test, y_test, verbose=0)
test_accuracy[1]
```

Out[103]:

```
0.7710360884666443
```

In [104]:
```
# Extract features from the audio file
filename = 'dog bark.wav'
prediction_feature = features_extractor(filename)
prediction_feature = prediction_feature.reshape(1, -1)

# Make predictions
predictions = model.predict(prediction_feature)
predicted_classes = np.argmax(predictions, axis=1)

print(predicted_classes)
```

```
1/1 [==============================] - 0s 110ms/step
[3]
```

In [113]:
```
metadata
```

Out[113]:

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
...	...	...	...	...	...	...	...	...
8727	99812-1-2-0.wav	99812	159.522205	163.522205	2	7	1	car_horn
8728	99812-1-3-0.wav	99812	181.142431	183.284976	2	7	1	car_horn
8729	99812-1-4-0.wav	99812	242.691902	246.197885	2	7	1	car_horn
8730	99812-1-5-0.wav	99812	253.209850	255.741948	2	7	1	car_horn
8731	99812-1-6-0.wav	99812	332.289233	334.821332	2	7	1	car_horn

8732 rows × 8 columns

In [106]:
```
# Filter metadata based on predicted class ID
predicted_class_id = predicted_classes[0]
filtered_metadata = metadata[metadata['classID'] == predicted_class_id]
# Load metadata

metadata = pd.read_csv('UrbanSound8K/metadata/UrbanSound8K.csv')

if len(filtered_metadata) == 0:
    print("Predicted class ID not found in the metadata.")
else:
    # Extract relevant columns from filtered metadata
    filtered_metadata = filtered_metadata[['classID', 'class']]

    # Display predicted class and class ID
    prediction = filtered_metadata.iloc[0]
    print(f"Predicted class: {prediction['class']}")
    print(f"Predicted class ID: {prediction['classID']}")

```

```
Predicted class: dog_bark
Predicted class ID: 3
```

In [107]:

```
import pandas as pd
from tabulate import tabulate

# Filter metadata based on predicted class ID
predicted_class_id = predicted_classes[0]
filtered_metadata = metadata[metadata['classID'] == predicted_class_id]

# Load metadata
metadata = pd.read_csv('UrbanSound8K/metadata/UrbanSound8K.csv')

if len(filtered_metadata) == 0:
    print("Predicted class ID not found in the metadata.")
else:
    # Extract relevant columns from filtered metadata
    filtered_metadata = filtered_metadata[['classID', 'class']]

    # Display first 10 rows of the filtered metadata in a table
    table_data = filtered_metadata.head(10).values.tolist()
    headers = filtered_metadata.columns.tolist()
    print(tabulate(table_data, headers, tablefmt='grid'))

```

```
+-----------+----------+
|   classID | class    |
+===========+==========+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
|         3 | dog_bark |
+-----------+----------+
```

Testing some audio data

- Preprocess the new audio data
- predict the class
- invere transform your predicted label

In [114]:

```python
# Specify the file path
filename = '21684-9-0-5.wav'

# Check if the file exists
try:
    open(filename)
except FileNotFoundError:
    print(f"File '{filename}' not found.")
    exit()

# Load the audio file
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')

# Extract MFCC features
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)

# Scale the features
mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)

# Print the scaled features
print(mfccs_scaled_features)

# Reshape the features
mfccs_scaled_features = mfccs_scaled_features.reshape(1, -1)

# Print the reshaped features
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)

# Make predictions using the model
predicted_probabilities = model.predict(mfccs_scaled_features)

# Get the predicted label with the highest probability
predicted_label = np.argmax(predicted_probabilities)

# Print the predicted label
print(predicted_label)

# Decode the predicted label using the labelencoder
prediction_class = labelencoder.inverse_transform([predicted_label])

# Print the predicted class
```

```
43 print(prediction_class)
```

```
[-178.82379     136.7633     -13.068221     48.14213      6.6529574
   36.10647     -12.541818    -3.522911    -12.262587    10.594036
   -4.1306596    14.425096    -2.3365233    16.041061     0.95959246
   13.80531      -1.9968798    5.126442     -2.5143142    4.8222094
   -6.7894983     2.2693586   -0.7009771     2.7691789   -0.5069647
   -0.89399356   -1.6918974    3.7261329     2.4733682   -1.0054693
   -3.8394966     0.2816141    6.8972626     2.6243734   -2.8658035
   -3.4202144    -1.4473431   -9.400884     -1.6816708    2.3436675 ]
[[-178.82379     136.7633     -13.068221     48.14213      6.6529574
    36.10647     -12.541818    -3.522911    -12.262587    10.594036
    -4.1306596    14.425096    -2.3365233    16.041061     0.95959246
    13.80531      -1.9968798    5.126442     -2.5143142    4.8222094
    -6.7894983     2.2693586   -0.7009771     2.7691789   -0.5069647
    -0.89399356   -1.6918974    3.7261329     2.4733682   -1.0054693
    -3.8394966     0.2816141    6.8972626     2.6243734   -2.8658035
    -3.4202144    -1.4473431   -9.400884     -1.6816708    2.3436675 ]]
(1, 40)
1/1 [==============================] - 0s 42ms/step
9
['street_music']
```

In [ ]:

```
1
```